Lecture 5
Cryptanalysis of Knapsack Cryptography

Instructor: Chris Peikert
Scribe: Eric Crockett

1 The Subset-Sum Problem

We begin by recalling the definition of the *subset-sum* problem, also called the "knapsack" problem, in its search form.

Definition 1.1 (Subset-Sum). Given positive integer weights $\mathbf{a} = (a_1, \ldots, a_n)$ and $s = \sum_{i=1}^{n} a_i x_i = \langle \mathbf{a}, \mathbf{x} \rangle \in \mathbb{Z}$ for some bits $x_i \in \{0,1\}$, find $\mathbf{x} = (x_1, \ldots, x_n)$.

The subset-sum problem (in its natural decision variant) is NP-complete. However, recall that NP-completeness is a *worst-case* notion, i.e., there does not appear to be an efficient algorithm that solves *every* instance of subset-sum. Whether or not "most instances" can be solved efficiently, and what "most instances" even means, is a separate question. As we will see below, there are highly structured subset-sum instances that are easily solved. Moreover, we will see that if the bit length of the a_i is large enough relative to n, subset-sum is easy to solve for almost every choice of $\mathbf{a}$, using LLL.

2 Knapsack Cryptography

Motivated by the simplicity and NP-completeness of subset-sum, in the late 1970's there were proposals to use it as the basis of public-key encryption schemes. In these systems, the public key consists of weights $\mathbf{a} = (a_1, \ldots a_n)$ chosen from some specified distribution, and to encrypt a message $\mathbf{x} \in \{0,1\}^n$ one computes the ciphertext

$$s = \mathsf{Enc}_{\mathbf{a}}(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle.$$

A major advantage of this kind of encryption algorithm is its efficiency: encrypting involves just summing up n integers, which is much faster than operations like modular exponentiation, as used in other cryptosystems. As for security, recovering the message $\mathbf{x}$ from the ciphertext is equivalent to solving the subset-sum instance $(\mathbf{a}, s)$, which we would like to be hard.[1] Of course, the receiver who generated the public key should have a way of decrypting the message. This is achieved by embedding a secret "trapdoor" into the weights, which allows the receiver to convert the subset-sum instance into an easily solvable one.

One class of easily solved subset-sum instances involves weights of the following type.

Definition 2.1. A *superincreasing sequence* $\mathbf{a} = (a_1, \ldots, a_n)$ is one where $a_i > \sum_{j=1}^{i-1} a_j$ for all i.

Given any superincreasing sequence $\mathbf{a}$ and $s = \langle \mathbf{a}, \mathbf{x} \rangle$, it is easy to find $\mathbf{x}$: observe that $x_n = 1$ if and only if $s > \sum_{j=1}^{n-1} a_i$. Having found x_n, we can then recursively solve the instance $(\mathbf{a}' = (a_1, \ldots, a_{n-1}), s' = s - a_n x_n)$, which still involves superincreasing weights.

Of course, we cannot use a superincreasing sequence as the public key, or it would be trivial for an eavesdropper to decrypt. The final idea is to embed a superincreasing sequence into a "random-looking" public key, along with a trapdoor that lets us convert the latter back to the former. The original method of doing so, proposed by Merkle and Hellman, works as follows:

1. Start with some superincreasing sequence $b_1, \ldots, b_n$.
2. Choose some modulus $m > \sum_{i=1}^{n} b_i$, uniformly random $w \leftarrow \mathbb{Z}_m^*$, and uniformly random permutation π on $\{1, \ldots, n\}$.

[1] We ignore the fact that accepted notions of security for encryption require much more than hardness of recovering the entire message. However, if the message *is* easy to recover by an eavesdropper, then the scheme is clearly insecure.

3. Let $a_i = w \cdot b_{\pi(i)} \bmod m$. The public key is $\mathbf{a} = (a_1, \ldots, a_n)$, and the trapdoor is (m, w, π).

The encryption of a message $\mathbf{x} \in \{0,1\}^n$ is then

$$s = \mathsf{Enc}_{\mathbf{a}}(\mathbf{x}) = \sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n} w \cdot b_{\pi(i)} x_i.$$

Given the trapdoor (m, w, π), we can decrypt s as follows: simply compute

$$s' = w^{-1} s = \sum_{i=1}^{n} b_{\pi(i)} x_i \bmod m,$$

and then solve the subset-sum problem for the (permuted) superincreasing b_i, treating s' as an integer in the range $\{0, \ldots, m-1\}$. This works because $\sum_{i=1}^{n} b_{\pi(i)} x_i < m$, so s' is the true subset-sum (not modulo anything).

It turns out that some care is needed in choosing the superincreasing sequence $b_1, \ldots, b_n$. For example, the natural choice of $b_i = 2^{i-1}$ ends up admitting some simple attacks. We won't discuss this issue in any detail, because it turns out that the Merkle-Hellman scheme (and almost all of its subsequent variants) can be broken using tools like LLL, regardless of what superincreasing sequence is used.

3 Lattice Attacks on Knapsack Cryptography

In 1982, Shamir showed how to break the basic Merkle-Hellman class of schemes. His attack used Lenstra's polynomial-time algorithm for fixed-dimension integer programming, which uses LLL as a subroutine. (Shamir's attack has been extended to break many subsequent versions of the Merkle-Hellman system.) Shortly thereafter, Lagarias and Odlyzko gave an incomparable attack (later simplified by Frieze) that solves almost all instances of "low-density" subset-sum problems.

Definition 3.1. The *density* of a subset-sum instance is $n / \max_i \log a_i$.

Frieze showed that if the a_i are uniformly random in $\{1, \ldots, X \geq 2^{n^2(1/2+\varepsilon)}\}$, corresponding to a density of about $2/n$ or less, then we can efficiently solve the subset-sum problem with very high probability over the choice of the a_i alone. We describe Frieze's algorithm in the remainder of these notes.

We are given a subset-sum instance $(\mathbf{a} = (a_1, \ldots, a_n), s = \langle \mathbf{a}, \mathbf{x} \rangle)$ for some $\mathbf{x} \in \{0,1\}^n$. Without loss of generality, we may assume that $s \geq (\sum_i a_i)/2$. If not, we replace s by $s' = (\sum_{i=1}^{n} a_i) - s$, then flip the bits of the answer $\mathbf{x}$ that we find. Note that this assumption implies $\mathbf{x} \neq \mathbf{0}$.

Let $B = \lceil \sqrt{n \cdot 2^n} \rceil$, and define a lattice $\mathcal{L}$ using the basis

$$\mathbf{B} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ -Ba_1 & -Ba_2 & \ldots & -Ba_n & Bs \end{pmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)}.$$

The attack simply runs LLL on this basis to obtain a nonzero lattice vector whose length is within a $2^{n/2}$ factor of $\lambda_1(\mathcal{L})$. The analysis below shows that with high probability, the obtained vector is of the form $\pm\binom{\mathbf{x}}{0}$, which reveals the solution $\mathbf{x}$.

Notice that for coefficient vector $\mathbf{z} = \binom{\mathbf{x}}{1}$, we have the nonzero lattice vector $\mathbf{Bz} = \binom{\mathbf{x}}{0} \neq \mathbf{0}$, which has norm at most $\sqrt{n}$. Also, any lattice vector has a final coordinate divisible by B, and if this coordinate is nonzero, then the vector has length at least $B > 2^{n/2} \cdot \|\mathbf{x}\| \geq 2^{n/2} \cdot \lambda_1(\mathcal{L})$. Therefore, LLL always yields a lattice vector whose final coordinate is zero, and in the remainder of the analysis we restrict our attention to such vectors.

We now show that with high probability, integer multiples of $\binom{\mathbf{x}}{0}$ are the *only* nonzero lattice vectors that can have length at most $2^{n/2}\sqrt{n} < B$. So LLL must return such a multiple, and since the returned vector is part of a basis, it must be $\pm\binom{\mathbf{x}}{0}$.

Consider an arbitrary nonzero vector $\binom{\mathbf{z}}{0} \in \mathbb{Z}^{n+1}$, where $\|\mathbf{z}\| < 2^{n/2}\sqrt{n}$ and $\mathbf{z}$ is not an integer multiple of $\mathbf{x}$. We want to bound the probability that this vector is in $\mathcal{L}$, i.e., the probability that $\binom{\mathbf{z}}{0} = \mathbf{B}\binom{\mathbf{z}}{z_{n+1}}$ for some $z_{n+1} \in \mathbb{Z}$. In such an event, we have

$$s \cdot |z_{n+1}| = |s \cdot z_{n+1}| = \left|\sum_{i=1}^{n} a_i \cdot z_i\right| \leq \|\mathbf{z}\| \sum_{i=1}^{n} a_i,$$

so $|z_{n+1}| \leq 2\|\mathbf{z}\|$ (recall that we assumed $s \geq (\sum_{i=1}^{n} a_i)/2$). So fix a particular such z_{n+1}. In order for $\binom{\mathbf{z}}{0}$ to be in $\mathcal{L}$, it must be the case that

$$\sum_{i=1}^{n} a_i z_i = z_{n+1} \cdot s = z_{n+1} \sum_{i=1}^{n} a_i x_i,$$

which implies that $\sum_{i=1}^{n} a_i y_i = 0$ where $y_i = (z_i - z_{n+1} x_i)$. Since $\mathbf{z}$ is not an integer multiple of $\mathbf{x}$, some $y_i \neq 0$, and we can assume that without loss of generality that $i = 1$. Therefore, we must have $a_1 = -(\sum_{i=2}^{n} a_i y_i)/y_1$.

With these observations, for any fixed $\mathbf{z}, z_{n+1}$ satisfying the above constraints, the probability that $\binom{\mathbf{z}}{0} \in \mathcal{L}$ is bounded by

$$\Pr_{a_i}\left[\sum_{i=1}^{n} a_i y_i = 0\right] = \Pr_{a_1}\left[a_1 = -(\sum_{i=2}^{n} a_i y_i)/y_1\right] \leq X^{-1},$$

because the a_i are chosen uniformly from $\{1, \ldots, X\}$. Finally, we apply the union bound over all legal choice of $\mathbf{z}, z_{n+1}$, of which there are at most

$$(2B+1)^n \cdot (4B+1) \leq (5B)^{n+1} \leq 2^{n^2(1/2+o(1))}.$$

Therefore, taking $X = 2^{n^2(1/2+\varepsilon)}$ for an arbitrarily small $\varepsilon > 0$, the probability that there exists any $\binom{\mathbf{z}}{0} \in \mathcal{L}$ satisfying the above constraints is at most $2^{-\Omega(n^2)}$, which is extremely small. This completes the analysis.

Variants. We showed that, except for integer multiples of $\binom{\mathbf{x}}{0}$, no lattice vector has length less than $2^{n/2}\sqrt{n}$. So, LLL's approximation factor of $2^{n/2}$ guarantees that it returns $\pm\binom{\mathbf{x}}{0}$. Inspecting the analysis, the $2^{n/2}$ factor accounts for the density bound of $2/n$.

What if we had an algorithm that achieves a better approximation factor, e.g., one that solves SVP *exactly*, or to within a $\text{poly}(n)$ factor? For a density of $\approx 1/1.6$ (i.e., the a_i have bit length $\approx 1.6n$), one can show (following the same kind of argument, but with tighter bounds on the number of allowed $\mathbf{z}$) that $\pm\binom{\mathbf{x}}{0}$ are the *only* shortest vectors in the lattice. Similarly, for density $1/\Theta(\log n)$, one can show that all lattice vectors not parallel to $\binom{\mathbf{x}}{0}$ are some $\text{poly}(n)$ factor longer than it. However, at densities above $2/3$ or so, $\binom{\mathbf{x}}{0}$ may no longer be a shortest nonzero vector in the lattice, so even an exact-SVP oracle might not reveal a subset-sum solution.